1. State Goals and Define the System

CPU: Intel® Xeon® Silver 4314 CPU @ 2.40GHz

Cores: 16

NIC: Mellanox Technologies MT2892 Family [ConnectX-6 Dx]

RAM: 131356700 kB Total (=131.3567 GB)

OS: Debian GNU/Linux 11

Kernel: 5.10.0-28-amd64 x86_64

NIC Driver: mlx5_core; Version 5.8-2.0.3

Trex version: 2.99 (Stateless)

Goals of the study:

Paul Raatschen has performed a study on whether the original Fail2ban process can be improved upon. In his study he determined that especially with many clients, Fail2ban struggled to keep up with the unwanted incoming data. To remedy this issue a more performant program, Simplefail2ban, was implemented and measured. An increase in performance was evident. Simplefail2ban also supported two modes of interprocess communication. In the “disk” approach, log messages were written into a logfile by the parsing program to then be read by simplefail2ban. The “shm” approach defined a memory segment that both the parsing program and simplefail2ban could access, which then was used for communicating log messages.

In this study a third option “sock” was implemented. This approach utilizes socket to establish communication between the parsing program and simplefail2ban. While Paul Raatschen has established that Fail2ban can be improved upon, it is unclear if the shared memory approach is ideal. Comparing the newly implemented socket approach to the shared memory and disk mode and evaluating its performance is the goal of this study.

2. List Services and Outcomes

Both implementations (socket vs. shared memory) provide the same service to the end-user. They receive a steady stream of wanted communication request and a significantly larger stream of unwanted request (simulating a DoS attack). They then parse the information of clients trying to disrupt the system to the intrusion prevention system (simplefail2ban). This service will then modify eBPF maps to preemptively drop messages from bad clients before they can reach the kernel and hog system resources.

Possible outcomes are that the software is either able to reliably parse log messages to the intrusion prevention system (simplefail2ban) or that it will become overwhelmed by the incoming communication requests. This would result in the intrusion preventions system not being able to ban the clients sending unwanted data, resulting in the system being unable to respond to valid communication requests and therefore making the services that the system provides unavailable for legitimate clients.

3. Select Metrics:

Choosing similar performance metrics according to the previous experiments by Paul Raatschen is vital to make direct comparisons possible (also because he did a good job selecting those criteria and using other metrics is non-sensical)

Performance metrics (directly quoted from Paul Raatschens bachelor thesis):

- Total amount of unwanted requests dropped (number of packets)
- Total amount of unwanted requests dropped, relative to the total amount of unwanted request sent (percentage)
- Amount of log messages processed by Simplefail2ban, relative [to] the amount of log messages sent by the test server (percentage)
- Central Processing Unit (CPU) utilization of Simplefail2ban (seconds of CPU time)

Higher is better for the first three metrics. The last metric should be minimized for the device under testing to be able to provide its services to valid clients.

4. List Parameters

Again, following in the footsteps, the parameters affecting performance will overlap heavily with the experiments performed by Paul Raatschen:

- CPU: 16 cores; no hyper-threading enabled
- NIC: MTU [(Maximum transfer unit)] 1500 bytes
- Trex: One interface, 30 threads

- Number of entries in eBPF maps for IPv4 & IPv6: 1000000
- Number of receiving threads used by the test server: 16
- Duration of measurement: 300 seconds
- Amount of valid traffic sent: 10000, 50000 PPS
- Number of clients sending valid traffic: 254

- Simplefail2ban parameters:
    - Number of banning threads used: 4 (shared memory [and socket]), 1 (logfile)
    - Number of hash table bins used: 6000011 *(Why exactly this size? I get that it is supposed to be a prime number, but why not any other arbitrary prime number instead?)*
    - Ban threshold for clients: 3
    - Ban time for clients: 30 Seconds
    - Line count for the shared memory buffer segments: 100000
    - Segment count for the shared memory buffer: 16
    - Buffer size for uring_getlines: 2048
    - Number of sockets: Same amount as the number of readers (= 1) *(This seems like an obvious bottleneck)*

5. Select Factors to Study

- Effects of differing amount of invalid traffic sent: 100k, 1m, 10m PPS
- Effects of differing number of clients sending invalid data: 65534, 131086
- IP stack: IPv4, IPv6, IPv4/IPv6 mixed

- Simplefail2ban factors:
    - IPC Type: Logfile/ Shared Memory/ Socket
    - Overwrite/ No Overwrite (shared memory only)
    - Enabling more 2nd Reader/ No 2nd Reader (shared memory and socket only)
    - Workload stealing/ No workload stealing (shared memory only)
    - Regex matching (With the current code “No Regex” matching is not possible)

6. Select Evaluation Technique

A realistic system is simulated. Using TRex, traffic is generated and sent to the DUT. The traffic can be modified to fit the outlined factors listed above. Measuring the number of packets that are dropped/passed via the eBPF maps is done with the xdp_ddos01_blacklist_cmdline program, originally provided by Florian Mikolajczak and later modified by Paul Raatschen (source code has unfortunately been lost).

7. Select Workload

Steady stream of both valid and invalid traffic from several clients according to the loads outlined above.

8. Design Experiments

Experiment 1: Replication of Simplefail2ban Logfile

While the testing environment is not entirely new, the versions of the software used differs. Therefore, a replication is necessary to validate and compare the previous results from Paul Raatschen to the new socket architecture.

Experiment 2: Replication of Simplefail2ban Shared Memory

While the testing environment is not entirely new, the versions of the software used differs. Therefore, a replication is necessary to validate and compare the previous results from Paul Raatschen to the new socket architecture.

Since the shared memory features many options that can modify performance, this measurement will once be conducted with nothing but the default options (Regex matching being an exception, because since the current implementation does not function properly without enabling it) and once with the best performing options as outlined in the bachelor thesis of Paul Raatschen.

Experiment 3: Simplefail2ban Sockets

Establishing baseline measurements to evaluate the performance. A variety of measurements will be conducted according to the factors outlined above. The exception being that only one udp_server and simplefail2ban application will be utilized. Using more instances of these applications will be explicitly measured in the following experiments.

Experiment 4: Replication of Simplefail2ban Shared Memory with 2nd Reader

While the testing environment is not entirely new, the versions of the software used differs. Therefore, a replication is necessary to validate and compare the previous results from Paul Raatschen to the new socket architecture.

Since the shared memory features many options that can modify performance, this measurement will once be conducted with nothing but the default options (Regex matching being an exception, because since the current implementation does not function properly without enabling it) and once with the best performing options as outlined in the bachelor thesis of Paul Raatschen.

Experiment 5: Simplefail2ban Sockets with 2nd Reader

To fairly compare performance to the shared memory approach, in this experiment only a second reader will be used.

Experiment 6: Sockets with n udp_server and m Simplefail2ban applications

In this experiment, the impact of multiple processes sending data to the IPS will be measured. Measuring the impact of adding multiple Simplefail2ban instances is of interest since the socket architecture will send all incoming data to all reading processes. This means that the receiving side of the potentially unwanted traffic will have to send the same data multiple times. This results in the differing instances of the Simplefail2ban application receiving all data, meaning that the IPS might have to recheck already processed log messages.

9. Analyze and Interpret Data

Not yet possible

10. Present Results

Not yet possible